①

[redacted]

COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

UMIACS–TR–90–4
CS–TR–2386

January 1990

Finding Minimum Width for Single–Layer Channel Routing in Linear Time

Ronald I. Greenberg*
Institute for Advanced Computer Studies and
Department of Electrical Engineering
University of Maryland
College Park, MD 20742, rig@umiacs.umd.edu

F. Miller Maley†
Department of Computer Science
Princeton University
Princeton, NJ 08544, fmm@princeton.edu

ABSTRACT

This paper provides a linear time algorithm for determining the minimum separation required to route a channel when the connections can be realized in one layer. It generalizes similar results for river routing by allowing single-sided connections. The approach can also be used to obtain a simplified routability test for single-layer switchboxes (routing in a rectangle).

*Work begun at the Massachusetts Institute of Technology with partial support by the Defense Advanced Research Projects Agency under Contract N00014–87–K–0825.

†Supported in part by a Mathematical Sciences Postdoctoral Research Fellowship from the National Science Foundation, grant DMS–8705835.

1 Introduction

In recent years, a good deal of attention has been given to the problem of planar or single-layer wire routing for VLSI chips. Especially popular has been river routing in the restricted sense of the term, i.e., the connection of two parallel rows of corresponding points[1] [17, 13, 10, 16, 4, 18]. Other works have considered routing within a rectangle [2], placement and routing within a ring of pads [1], or routing between very general arrangements of modules [12, 9, 3].

These works have considered various wiring models (e.g. rectilinear or general) and several specific questions. Generally, there is some tradeoff between the generality of the wiring patterns considered and the sophistication of the questions asked. Asking for the best placement or routing of complicated arrangements of modules or wires would require solution of an NP-complete problem [14, 7, 8, 11, 6], but relatively sophisticated questions can be efficiently answered in the case of river routing. (See [13] for a particularly exhaustive list of specific problems in river routing.) In particular, the *minimum separation problem* [17, 13, 10, 16, 4] involves determining the minimum separation for two horizontal rows of terminals (in fixed horizontal positions) which will allow the routing to be performed. This problem can be solved in linear time under the rectilinear and most other wiring models [13, 10, 16, 4], less time than is required to perform the complete routing.

This paper concentrates on showing that the minimum separation problem (in the rectilinear wiring model) can be solved in linear time for any single-layer channel routing problem, even with single-sided connections. In the process, this paper clearly provides a linear-time solution to the *routability problem* which simply asks whether a routing is possible given a fixed vertical separation as well as fixed horizontal positions of the terminals. We also briefly discuss generalization from channels to switchboxes, demonstrating, in particular, a simple routability test for switchboxes. Comparable results could be obtained fairly directly from the work of Cole and Siegel [3], but this paper gives a simpler exposition by concentrating on the special cases of channels and switchboxes. Detailed but concise pseudocode is provided for the channel separation problem.

One of the motivations for considering the minimum separation problem for single-layer channels is recent work on a heuristic multi-layer channel router, MulCh [5], which finds good solutions to general channel routing problems by dividing them into essentially independent subproblems of one, two, or three layers. A main step in MulCh is to greedily partition the nets once a set of layer groups has been determined. As each net is considered, it is assigned to the group where the resulting subproblem seems to be the one requiring the least separation. For two and three layer subproblems, density and length of the longest VCG[2] path provide good measures of routing difficulty, but for a single-layer problem, these can be very poor measures. Since MulCh must consider a large number of potential net assignments, it settles on the density measure, an easy linear time calculation, to evaluate

[1] This is the only usage of the term "river routing" in this paper; more complicated variations of the problem are referred to as "single-layer" or "planar" routing.

[2] The vertical constraint graph is formed by using a node to represent each net and including an edge from net A to net B if a pin for net A appears above a pin for net B in some column of the channel.

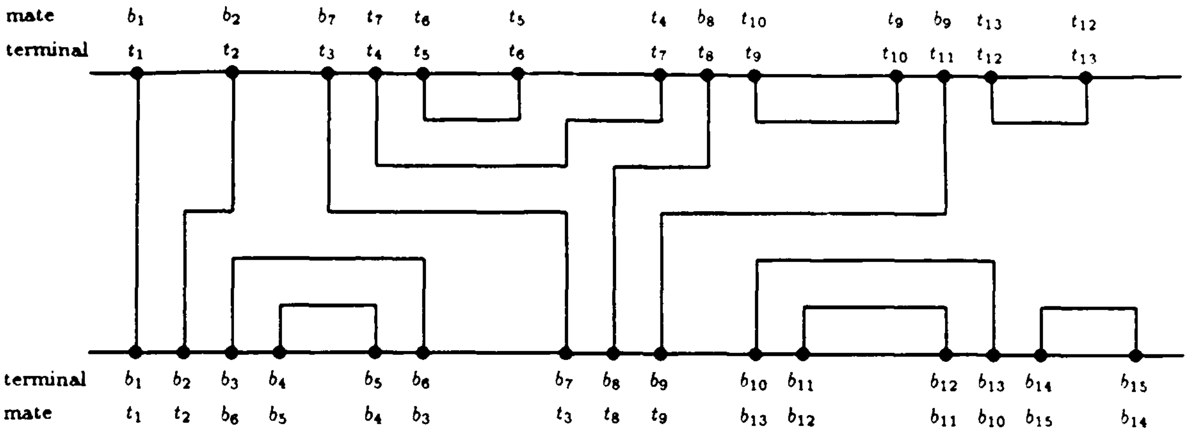
mate
terminal
b1 b2 b7 t7 t6 t5 t4 b8 t10 t9 b9 t13 t12
t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t13
terminal
mate
b1 b2 b3 b4 b5 b6 b7 b8 b9 b10 b11 b12 b13 b14 b15
t1 t2 b6 b5 b4 b3 t3 t8 t9 b13 b12 b11 b10 b15 b14

Figure 1: A representative single-layer channel routing problem. Each terminal T among the top terminals $\{t_i\}$ or the bottom terminals $\{b_i\}$ must connect to exactly one terminal $mate(T)$.

assignments to single-layer groups. Incorporating the linear time minimum separation algorithm of this paper instead, would provide MulCh with a greater ability to make good net partitions.

The remaining sections of this paper provide a more detailed definition of the single-layer channel routing problem, the algorithm for finding minimum channel separation, and discussion of the generalization to switchboxes.

2 Problem Definition and Notation

Figure 1 illustrates a single-layer channel routing problem and some of the notation to be used. The terminals along the top and bottom are labeled $t_1, t_2, \ldots, t_m$ and $b_1, b_2, \ldots, b_n$, respectively. In river routing, the problem would be to connect t_i to b_i for every i; in our more general channel routing problem we allow single-sided connections as between b_3 and b_6 in Figure 1.[3] For each each terminal T, we use the notation mate(T) to represent the terminal which is to be connected to T.[4] We will also make the notation for terminals do double duty; where a terminal appears in an arithmetic context (comparison or subtraction), it will represent the x-coordinate or horizontal position of the terminal.

A few details remain in order to fully define the problem. First, in this paper, we assume a rectilinear wiring model in which all wires lie on the lines of an underlying grid and the terminals on grid points. Secondly, though wires are required to stay unit distance apart, it is convenient to assume that wires may be routed immediately alongside the channel boundaries. If practical considerations require that wires not run along the channel

[3] We use the term "single-sided" for these connections even though the term "sides" is sometimes used to refer to the left and right of the channel as opposed to the top and bottom where the terminals lie.

[4] This restriction to two-point nets represents no loss of generality in the case of single-layer routing, since it is easy to transform problems with multiterminal nets to problems with only two terminal nets (but a larger number of nets) and to perform the reverse transformation on the routings obtained.

boundaries (except perhaps when connecting terminals on adjacent grid points along the top or bottom of the channel), the results in this section can be applied by simply routing wires one unit into the channel from each terminal, determining the minimum separation of the new rows of terminals, and adding two to obtain the separation of the original channel boundaries. The only case where this will not yield the optimal result is when all the nets are trivial in the sense that each net simply runs straight across the channel or connects adjacent grid points on the channel boundary. It is easy to check for this special case, so this paper assumes that routing along channel boundaries is permitted.

3 Finding Minimum Channel Separation

This section shows how to determine minimum separation for the channel routing problem just described. The first step in the demonstration is the invocation of a general theory of single-layer routing [12, 3] specialized to channel routing in order to obtain a set of necessary and sufficient conditions for routability. Then it is shown that we can reduce the number of conditions which must be checked, and, finally, that the minimum channel width satisfying the conditions can be determined in linear time.

It is most convenient for us to use the single-layer routing theory of [12] in a slightly altered form consistent with the approach of Cole and Siegel [3]. The basic idea of the theory is that routability can be decided by checking whether a limited collection of *cuts* are *safe*. Roughly, a cut is a line segment from top to bottom of the channel, and it is safe if there is enough room for all the nets which must cross it to do so. The alterations to Maley's theory were foreshadowed when it was indicated that we will allow routing along channel boundaries. It is convenient to have the terminals and the two channel boundaries play the role of *features* in the theory of [12]. Though this represents a change in that wires were required to stay unit distance away from nonterminal features in [12], it is only necessary to slightly alter a few definitions by considering cuts as closed rather than open line segments. The key definitions which place the theory in the form we desire are summarized below.

Definition: A *critical cut* is a line segment connecting a terminal on the top of the channel to a terminal on the bottom or a line segment running from a terminal straight across to the other channel boundary.

Definition: The *flow* of a cut c, denoted flow(c), is the number of wires which *must* cross c, including the wires incident at an endpoint of c. More precisely, the flow is the number of nets that have terminals lying on opposite sides of c or that have a terminal coincident with an endpoint of c.

Definition: The *capacity* of a cut c from b to t, denoted cap(c), is $\|b - t\| + 1$, where $\|\cdot\|$ represents the ℓ_∞ norm, i.e., $\|(x, y)\| = \max\{|x|, |y|\}$.

Definition: A cut c is *safe* if an only if flow(c) $\leq$ cap(c).

We can now state formally, the result which we need from the general theory of single-layer routing.

Lemma 1 *A channel is routable if and only if all of the critical cuts are safe.*

Proof. The lemma follows immediately from the corresponding result in [12, p. 40] using slightly different definitions. ■

The restriction to checking critical cuts gives us $O(N^2)$ conditions which must be checked, where $N = m + n$ is the number of terminals. In the river routing problem, it can be shown that it suffices to check a smaller set of $O(N)$ cuts, but when single-sided connections are allowed it is not evident how to reduce the number of cuts below $\Omega(N^2)$. Nonetheless, it is possible to eliminate some of the critical cuts in a manner dependent on the flows and capacities of the cuts, leaving a set of cuts for which the flows and capacities are sufficiently nicely related that linear time suffices to determine the minimum channel width that makes all the cuts safe.

The necessary limitation on the critical cuts to be checked is provided by the simple distinction between *dense* and *sparse* cuts. A sparse cut is one which is safe regardless of the channel separation, i.e., the horizontal distance between the cut endpoints is large enough relative to the flow that the cut is guaranteed to be safe. Any cut which is not sparse is referred to as a dense cut. For convenience, we also classify as dense any cut running from a terminal straight across the channel. (Such a cut is both sparse and dense if its flow is 1.) Thus, any channel routing problem has dense cuts, and, in fact, any terminal is the endpoint of some dense cut. With the distinction between dense and sparse cuts, we can provide a simple expression for the channel separation as given in the following corollary to Lemma 1.

Corollary 2 *Minimum channel separation is given by* $-1 + \max_{c \text{ dense}} \text{flow}(c)$. ■

The next key observation is that the set of dense cuts emanating from any given terminal form a "cone". In Figure 2, for example, the dotted and dashed lines show all the critical cuts emanating from terminal a on the bottom of the channel. Terminal l is the leftmost terminal on the top for which the cut from a is dense, and terminal r is the rightmost such terminal. The critical cuts to points between l and r inclusive are all dense, while the cuts outside of this cone are all sparse. To see that this situation prevails in general, just compare any sparse cut to a cut "immediately below" it. For example, in Figure 2, the sparsity of cut $[a, s]$ implies that cut $[a, s']$ must be sparse. This is true because the horizontal extent of $[a, s']$ is greater than that of $[a, s]$, but the flow of $[a, s']$ can be at most one greater than that of $[a, s]$. (In this example the flows are the same.)

The basic idea of the algorithm for finding minimum channel separation is to sweep a cone of dense cuts across the channel, moving the apex of the cone along the bottom terminals. As we sweep across the channel, we compute the maximum of the flows of the

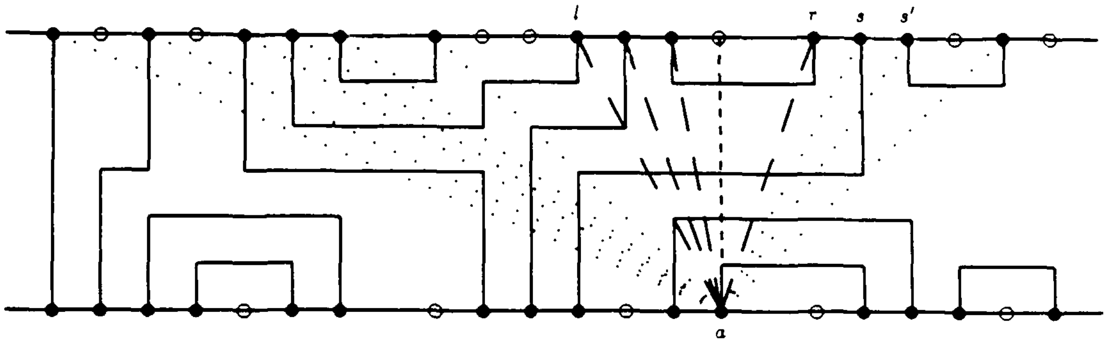

Figure 2: The dotted and dashed lines show the sparse and dense cuts, respectively, emanating from terminal a. In general, the dense cuts emanating from any terminal form a "cone". This figure also shows the dummy terminals (hollow circles) which are added for convenience in detailing the algorithm.

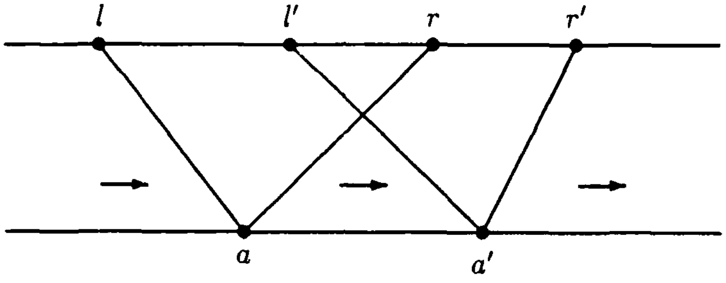

Figure 3: The dense cuts are checked by sweeping across the channel from one bottom to terminal to the next, maintaining a cone of dense cuts incident to the bottom terminal. At each stage, the algorithm keeps track of the maximum of the flows of all dense cuts seen so far.

dense cuts. By marching the apex of the cone only along the bottom, we may miss straight cuts emanating from top terminals, but this omission can be remedied by adding dummy bottom terminals at x-positions which have a top terminal but no bottom terminal. (For any dummy terminal T, mate(T) will have a special null value.) In fact, for simplicity of coding the algorithm, it is convenient to also add dummy top terminals at x-positions which have a bottom terminal but no top terminal. The positions of the dummy terminals are illustrated in Figure 2.

The cone sweeping approach is illustrated in Figure 3 which shows the cone of cuts as we move the apex from one bottom vertex to the next. In the figure, l and r are the leftmost and rightmost top terminals for which the cut from a is dense, and l' and r' play the same roles for the next bottom terminal a'. The main loop of the algorithm involves moving the apex of the cone through the bottom terminals from left to right. At each stage, we update the leftmost and rightmost top terminals corresponding to dense cuts, and, as will be explained further below, we maintain a collection of the flow values of cuts within the cone. The maximum of these flow values is compared to the maximum seen so far so that the global maximum is known at the end of the sweep.

The first step in showing that the channel separation algorithm runs in linear time is to argue that each of the top boundaries of the cone of dense cuts moves monotonically across the channel. That is, with reference to Figure 3, l' is at least as far right as l, and r' is at least as far right as r. This a consequence of the fact that thc dense cuts emanating from any terminal form a cone (as applied to a top terminal rather than a bottom terminal). This assures us that there are only a linear number of steps involved in moving the leftmost and rightmost top terminals of interest as the apex of the cone moves along the bottom terminals.

Now we simply need to show that we can maintain the maximum of the flows of the dense cuts without exceeding linear time as the cone sweeps across the channel.

First of all, note that each step of pushing the rightmost top terminal one terminal to the right is a constant time operation. It merely requires computing the flow of a new cut whose flow differs by -1, 0, or 1 from the flow of the "preceding" cut. Let us define a function flowdiff to represent this difference:

$$\text{flowdiff}(p, q, r) = \text{flow}([p, q]) - \text{flow}([p, r]) .$$

When q and r are adjacent terminals on one of the channel boundaries, the value of flowdiff(p, q, r) can be computed by checking on which sides of the cuts $[p, q]$ and $[p, r]$ the mates of q and r lie. As the rightmost top terminal is moved right, each new flow is associated to the relevant terminal and tallied in a table of flows in the current collection of cuts as illustrated in Figure 4.

The second operation we must treat is that of moving the leftmost top terminal one terminal to the right. This operation can be performed by looking up the already computed flow associated with the terminal being dropped from the cone at the left. Once we know this flow, we simply decrement the appropriate tally in the table of flows.

Finally, though it is less obvious, constant time also suffices to move the apex of the cone one terminal to the right while retaining the correct values of the flows of the cuts to the top terminals. This is true because when the apex of the cone moves one terminal to the right, the flows of cuts to relevant top terminals all change by the same amount (0, 1, or -1). That is, for all top terminals which are in the cone of dense cuts for both the old and the new apex, the change in flow is the same. A straightforward case analysis shows that if this were false, there would have been a cut of unit flow at the borderline between the cuts which experience different flow changes. But this cut and those below it would be sparse and therefore cannot be in the cone of dense cuts for both the old and new apex. Thus, when the apex of the cone moves, it is only necessary to adjust a flow offset which will be applied to all the flows in the table once the upper range of the cone is updated.

The only trick that remains is to keep track of the maximum flow of cuts in the table. Then after each movement of the apex and the appropriate adjustment of the leftmost and rightmost top terminals, this maximum flow plus the flow offset is compared to the running maximum over the whole channel. It is easy to update the maximum of flows in the table when a new flow is added; just compare the new flow to the old maximum. Fortunately, it is also unnecessary to do any searching when a flow is removed. Since adjacent cuts differ by flow at most one, the flow values which have nonzero tallies in the table will always be a continuous range of integers. Thus, to update the maximum flow in the table, it is

	flow	#cuts
	⋮	
	5	0
	4	0
maxflow →	3	2
	2	2
	1	1

offset 3

Figure 4: The table used to tally the flows in the current collection of dense cuts.

only necessary to subtract 1 from the maximum if the flow being removed equals the old maximum and is the last flow of that value in the collection.

The arguments provided above suffice to show that the procedure in Figure 5 computes the minimum channel separation in linear time. Below is a more detailed guide to the code in Figure 5.

The variables in Figure 5 are used as follows. The variable *apex* represents the index among the bottom terminals of the apex of the cone of cuts. The variables *left* and *right* are used to keep track of the indices among the top terminals of the leftmost and rightmost endpoints of dense cuts emanating from the apex of the cone. The array *flow*, as modified by the variable *offset* associates to top terminals delimited by *left* and *right* the flow of the cuts to these terminals from the apex of the cone. The array *tally* corresponds to the table in Figure 4; for each possible value of *flow*, it keeps track of the number of top terminals delimited by *left* and *right* which correspond to that value. The variable *maxflow* is the maximum index of *tally* for which the value of the array element is nonzero. That is, *maxflow* represents the maximum value of *flow* corresponding to a terminal delimited by *left* and *right*. Finally, *separation* is the running maximum (minus 1) of the flows of all dense cuts in the channel.

The initializations for procedure CHANNEL-SEPARATION are found in lines 1–6. Line 1 is used to conveniently ensure that straight cuts will be checked and the cone of dense cuts will never be empty. Initially, the apex of the cone is at the leftmost bottom terminal and the single cut in the cone is $[b_1, t_1]$. (The cone will be expanded later if necessary.) Lines 5–6 initialize the flow collection to contain just the information for the cut $[b_1, t_1]$. Note that a range of $-n$ to $n + \frac{1}{2}(m+n)$ for the indices of *tally* is certainly adequate since the flow of any cut is in the range $[1, \frac{1}{2}(m+n)]$, and the offset is in the range $[-n, n]$. Furthermore, for the cuts we consider at a given position of the apex, flow minus offset is upper bounded by $\frac{1}{2}(m+n)$. Hence the range of indices used in line 6. (Additional analysis may restrict the necessary range of indices further, but the range used is certainly sufficient.)

The main loop of procedure CHANNEL-SEPARATION, comprising the cone sweeping operation, is found in lines 7–26. There are four main parts to this loop. Lines 8–16 move the rightmost top terminal under consideration towards the right until the *next* cut from the apex would be sparse (and directed towards the right from the apex). Similarly, Lines 17–23 move the leftmost top terminal under consideration towards the right until the

```
procedure CHANNEL-SEPARATION
Add dummy terminals (with null mates) so that every $x$-position with a
  terminal has both a top and bottom terminal, yielding top and bottom
  terminal sets $\{t_1, t_2, \ldots, t_{last}\}$ and $\{b_1, b_2, \ldots, b_{last}\}$.
$apex \leftarrow 1$
$left \leftarrow 1$ ; $right \leftarrow 1$
$offset \leftarrow 0$ ; $flow(1) \leftarrow \mathrm{flow}([b_1, t_1])$
    $\langle offset + flow(right) = \mathrm{flow}([b_{apex}, t_{right}])$ always$\rangle$
$maxflow \leftarrow flow(1)$
$tally(i) \leftarrow 0$ for $-n \le i \le \frac{1}{2}(m+n)$ ; $tally(flow(1)) \leftarrow 1$
while $apex \le last$
    while $right < last$
        $flow(right+1) \leftarrow flow(right) + \mathrm{flowdiff}(b_{apex}, t_{right+1}, t_{right})$
        if $t_{right+1} - b_{apex} \ge \max\{1, flow(right+1) + offset - 1\}$ then
            break    $\langle sparse\ cut\rangle$
        endif
        $right \leftarrow right + 1$
        $tally(flow(right)) \leftarrow tally(flow(right)) + 1$
        $maxflow \leftarrow \max\{maxflow, flow(right)\}$
    endwhile
    while $b_{apex} - t_{left} \ge \max\{1, flow(left) + offset - 1\}$    $\langle sparse\ cut\rangle$
        $tally(flow(left)) \leftarrow tally(flow(left)) - 1$
        if $maxflow = flow(left)$ and $tally(flow(left)) = 0$ then
            $maxflow \leftarrow maxflow - 1$
        endif
        $left \leftarrow left + 1$
    endwhile
    $separation \leftarrow \max\{separation, maxflow + offset - 1\}$
    $apex \leftarrow apex + 1$ and $offset \leftarrow offset + \mathrm{flowdiff}(t_{right}, b_{apex}, b_{apex-1})$
endwhile
```

Figure 5: This algorithm computes the maximum of the flows of the dense cuts, thereby determining the minimum channel separation for which all cuts are safe. Comments appear inside angle brackets. Variable names are set in italics; function names appear in roman type.

cut running from it to the apex is *not* sparse (or it is not to the left of the apex). The dense cuts then correspond to the top terminals which are at least as far right as *left* and at least as far left as *right*. Line 24 updates the running maximum of the flows of the dense cuts. Finally, line 25 moves the apex of the cone one terminal to the right along the bottom terminals and updates the offset to the table of flows being maintained. Within the loop that updates *right*, the main steps are computation of the new flow and updating of the flow collection and *maxflow* illustrated in Figure 4. Within the loop that updates *left* the operations involved are to update the flow collection by looking up the flow value for the cut being dropped from consideration and then to update *maxflow* by taking advantage of the contiguous nature of the nonzero values in the flow collection as described above.

4 Switchbox Routability Testing

This section considers extension of the above results from channels to switchboxes. That is, terminals are allowed at the sides of the channel instead of just at the top and bottom. With floating side-terminals, the problem is shown to easily reduce to the problem without side-terminals. When side-terminals are fixed, it is shown how to obtain a linear time routability test which is somewhat simpler than that of Chang and JáJá [2].[5]

The first variant of switch box routing, in which side-terminals are allowed to float to arbitrary y-positions may be thought of as merely an extended channel (figuratively and literally). (Included in this notion of floatability is that there are no single-sided side connections.) It suffices to extend the channel, place the side terminals along the top or bottom of the channel, and solve the problem as before. As long as there are no single-sided side connections, the wires attached to the former side terminals must cross the former channel sides, so any routing for the new problem can be converted to a routing for the old problem by merely truncating some wires.

When the side-terminals are fixed, a linear time routability test is obtained with little more than two applications of the procedure CHANNEL-SEPARATION. After applying that procedure to the terminals at top and bottom of the switchbox and then applying it with the appropriate shift in orientation to the terminals at left and right of the switchbox, it is only necessary to check for safety of the cuts which connect perpendicular sides of the switchbox. Fortunately, checking these corner cuts is as easy as for river routing without side connections [3]; we need only check the diagonal cuts, which is a simple, linear time procedure.

5 Conclusion

We have demonstrated simple linear-time algorithms for several single-layer routing problems. Specifically, we can determine minimum width for channels or channels with floating side-terminals, and we can test routability in a switchbox with all terminal positions fixed.

[5] Pinter also considers this problem [14, 15], but his linear time routability test contains a bug.

Further directions for research involve minimizing channel width or switchbox area with other types of constraints on terminal positioning.

References

[1] Brenda S. Baker and Ron Y. Pinter. An algorithm for the optimal placement and routing of a circuit within a ring of pads. In *24th Annual Symposium on Foundations of Computer Science*, pages 360–370. IEEE Computer Society Press, 1983.

[2] Shing-Chong Chang and Joseph JáJá. Optimal parallel algorithms for river routing. Unpublished manuscript, 1988.

[3] Richard Cole and Alan Siegel. River routing every which way, but loose. In *25th Annual Symposium on Foundations of Computer Science*, pages 65–73. IEEE Computer Society Press, 1984.

[4] Danny Dolev, Kevin Karplus, Alan Siegel, Alex Strong, and Jeffrey D. Ullman. Optimal wiring between rectangles. In *Proceedings of the 13th ACM Symposium on Theory of Computing*, pages 312–317. ACM Press, 1981.

[5] Ronald I. Greenberg, Alexander T. Ishii, and Alberto L. Sangiovanni-Vincentelli. MulCh: A multi-layer channel router using one, two, and three layer partitions. In *IEEE International Conference on Computer-Aided Design (ICCAD-88)*, pages 88–91. IEEE Computer Society Press, 1988.

[6] Gershon Kedem and Hiroyuki Watanabe. Graph-optimization techniques for IC layout and compaction. In *Proceedings of the 20th ACM/IEEE Design Automation Conference*, pages 113–120. IEEE Computer Society Press, 1983.

[7] Mark R. Kramer and Jan van Leeuwen. Wire-routing is NP-complete. Technical Report RUU-CS-82-4, University of Utrecht, the Netherlands, February 1982.

[8] Andrea Suzanne LaPaugh. *Algorithms for Integrated Circuit Layout: An Analytic Approach.* PhD thesis, Department of Electrical Engineering & Computer Science, Massachusetts Institute of Technology, November 1980. MIT/LCS/TR-248.

[9] Charles E. Leiserson and F. Miller Maley. Algorithms for routing and testing routability of planar VLSI layouts. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, pages 69–78. ACM Press, 1985.

[10] Charles E. Leiserson and Ron Y. Pinter. Optimal placement for river routing. *SIAM Journal on Computing*, 12(3):447–462, August 1983.

[11] T. Lengauer. On the solution of inequality systems relevant to IC-layout. *Journal of Algorithms*, 5(3):408–421, September 1984.

[12] F. Miller Maley. *Single-Layer Wire Routing.* PhD thesis, Department of Electrical Engineering & Computer Science, Massachusetts Institute of Technology, August 1987. MIT/LCS/TR-403.

[13] Andranik Mirzaian. River routing in VLSI. *Journal of Computer and System Sciences*, 34:43–54, 1987.

[14] Ron Y. Pinter. River routing: Methodology and analysis. In Randal Bryant, editor, *Third Caltech Conference on Very Large Scale Integration*, pages 141–163. Computer Science Press, March 1983.

[15] Ron Yair Pinter. *The Impact of Layer Assignment Methods on Layout Algorithms for Integrated Circuits.* PhD thesis, Department of Electrical Engineering & Computer Science, Massachusetts Institute of Technology, August 1982. MIT/LCS/TR-291.

[16] Alan Siegel and Danny Dolev. The separation for general single-layer wiring barriers. In H. T. Kung, Bob Sproull, and Guy Steele, editors, *VLSI Systems and Computations*, pages 143–152. Computer Science Press, 1981.

[17] Alan Siegel and Danny Dolev. Some geometry for general river routing. *SIAM Journal on Computing*, 17(3):583–605, June 1988.

[18] Martin Tompa. An optimal solution to a wire-routing problem. *Journal of Computer and System Sciences*, 23:127–150, 1981.